Application of CMSA to the Minimum Capacitated Dominating Set Problem

Pedro Pinacho-Davidson
Department of Computer Science,
Universidad de Concepción
Concepción, Chile
ppinacho@udec.cl

Salim Bouamama
Department of Computer Science,
University of Ferhat Abbas - Sétif 1
Sétif, Algeria
salim.bouamama@univ-setif.dz

Christian Blum
Artificial Intelligence Research
Institute (IIIA-CSIC)
Bellaterra, Spain
christian.blum@iiia.csic.es

ABSTRACT

This work deals with the so-called minimum capacitated dominating set (CAPMDS) problem, which is an NP-Hard combinatorial optimization problem in graphs. In this paper we describe the application of a recently introduced hybrid algorithm known as Construct, Merge, Solve & Adapt (CMSA) to this problem. Moreover, we evaluate the performance of a standalone ILP solver. The results show that both CMSA and the ILP solver outperform current state-of-the-art algorithms from the literature. Moreover, in contrast to the ILP solver, the performance of CMSA does not degrade for the largest problem instances. The experimental evaluation is based on a benchmark dataset containing two different graph topologies and considering graphs with variable and uniform node capacities.

CCS CONCEPTS

• **Theory of computation → Discrete optimization**;

KEYWORDS

Minimum capacitated dominating set; hybrid algorithm; construct, merge, solve & adapt

ACM Reference Format:
Pedro Pinacho-Davidson, Salim Bouamama, and Christian Blum. 2019. Application of CMSA to the Minimum Capacitated Dominating Set Problem. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3321707.3321807

1 INTRODUCTION

Dominating set problems are hard combinatorial optimization problems from the family of set covering problems that have recently become a subject of interest due to their application in both clustering and routing in wireless networks [11, 15, 18]. The best known dominating set problem is the minimum dominating set (MDS) problem. Variants of the MDS problem include the minimum connected dominating set (MCDS) problem and the minimum capacitated dominating set (CAPMDS) problem. Generally, these problem variants may be solved in both unweighted and weighted graphs. In this paper, we focus on the CAPMDS problem. Remember that, given an undirected graph G with vertex set V (and $|V| = n$), any set $D \subseteq V$ is called a dominating set iff each vertex $v \in V$ is either in D or has at least one neighbor that is a member of D. The MDS problem requires to find a dominating set of minimum cardinality. In addition, the CAPMDS problem limits the number of vertices that each vertex can dominate (a more detailed description will be given in the next section). Due to the importance of the problem in the context of applications in wireless communication, several approaches have been proposed in the literature in recent years. Even though the CAPMDS problem has been proven to be NP-hard [4], some researchers have focused on the development of exact approaches. Cygan et *al.* [3] presented an algorithm based on maximum matching that runs in time $O(1.89^n)$. The performance of the algorithm was further improved by considering dynamic programming over subsets [8] with a time complexity of $O(1.8463^n)$. A distributed approximation scheme has been developed in [5]. This algorithm achieves a $O(\log\triangle)$-approximation in $O(\log^3 n + \log(n)/\epsilon)$ time, where n represents the number of vertices and $\triangle$ denotes their maximal degree. Finally, the CAPMDS problem has been subject to a few research studies focused on heuristics and metaheuristic approaches. Potluri and Singh [13] carried out a performance comparison between three greedy heuristics, a main one and two variations. These algorithms generate a solution step by step, adding one vertex at a time. They showed that the best heuristic is the one that selects, at each step, the vertex that maximizes the minimum between the vertex capacity and the number of uncovered neighbors. The neighboring vertices dominated by this vertex (in case the neighborhood is larger than the capacity of the vertex) are chosen randomly. This greedy heuristic has been used as the basis for the design of two metaheuristic approaches, one based on ant colony optimization (named ACO) and the other one based on genetic algorithms (named SGA) [14]. Recently, Li et *al.* [7] developed an iterated local search approach labelled LS_PD with a significantly better performance than ACO and SGA when applied to general graphs with uniform and variable capacity. LS_PD adopts a penalization strategy in the context of the vertex-scoring scheme. Moreover, it makes use of a two-mode dominated vertex selection strategy taking into account both random and greedy decisions for the choice of the neighbors that a chosen vertex should dominate. This is done for the purpose of achieving a balance between the intensification and the diversification of the search process.

The aim of this study is to propose a new approach for the CAPMDS problem based on a recently proposed hybrid algorithm for combinatorial optimization labelled Construct, Merge, Solve, & Adapt (CMSA) [2]. This algorithm combines an exact solver with heuristic concepts to accelerate the solution process in order to be

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321807

able to deal with large-scale problem instances.

The remainder of the paper is organized as follows. Section 2 formally describes the CAPMDS problem in more detail and presents an integer linear programming (ILP) model for the considered problem. Section 3 presents the proposed algorithm. Section 4 is dedicated to the experimental results. Finally, section 5 summarizes this work, draws conclusions, and outlines possible future work.

2 THE CAPMDS PROBLEM

Before describing the CAPMDS problem, let us briefly recall some relevant definitions and notions. Let $G = (V, E)$ be an undirected graph on a set of n vertices $V = \{v_1, v_2, \cdots, v_n\}$ and a set of edges E. Moreover, we assume that the graph is simple, that is, it neither contains loops nor multi-edges. Two vertices are neighbors (adjacent to each other) if and only if there exists an edge between them, that is, $v \in V$ and $u \in V$ are said to be neighbors if $(v, u) \in E$. For a vertex v, let $N(v) := \{u \in V \mid (v, u) \in E\}$ denotes the set of neighbors, known as the *open neighborhood*, of v in G. Besides, the *closed neighborhood* of a vertex v in G, denoted by $N[v]$, contains the vertices adjacent to v plus vertex v itself, that is, $N[v] := N(v) \cup \{v\}$. The degree $\deg(v)$ of v is the cardinality of the set of neighbors of v, that is, $\deg(v) = |N(v)|$. A dominating set is a subset $S \subseteq V$ such that each vertex $v \in V \setminus S$ must be adjacent to at least one vertex in S. Each vertex in S is called a *dominator*, otherwise it is called a *dominatee*. A dominator dominates (covers) itself and (some of) its neighbors.

A problem instance of the CAPMDS problem is a tuple (G, Cap) that consists of an undirected (simple) graph $G = (V, E)$ and a capacity function $Cap : V \to \mathbb{N}$ that associates a positive number $Cap(v)$ to each vertex $v \in V$ representing the maximum number of adjacent vertices this vertex is allowed to dominate. A subset $S \subseteq V$ is said to be a capacitated dominating set if there exists a domination mapping (also called dominating function) $f_S : V \setminus S \to S$ such that the following holds:

(1) Each vertex $u \in V \setminus S$ is mapped to one of its neighbors v from S, that is $f_S(u) = v$ with $v \in N(u) \cap S$.
(2) $|f_S^{-1}(v)| \leq Cap(v)$ for each vertex $v \in S$, where $f_S^{-1}(v) := \{u \in V \setminus S | f_S(u) = v\}$. In other words, $|f_S^{-1}(v)|$ denotes the total number of vertices mapped by f_S to a single vertex $v \in S$. Obviously, $|f_S^{-1}(v)|$ must not be greater than $Cap(v)$, the capacity of v.

Finally, a minimum capacitated dominating set is a capacitated dominating set with minimum cardinality.

Figure 1 provides an illustrative example of the CAPMDS problem. While Figure 1a shows an example graph, Figure 1b shows an optimal solution (black vertices) assuming a uniform capacity of 2 for each vertex. Note that the assignments of the domination mapping are shown in terms of bold edges. Vertices v_5 and v_9, for example, are mapped to vertex v_6, which forms part of the solution. More specifically, $S = \{v_2, v_3, v_6, v_7, v_{12}\}$ and $f_S^{-1}(v_2) = \{v_1, v_4\}$, $f_S^{-1}(v_3) = \{v_8\}$, $f_S^{-1}(v_6) = \{v_5, v_9\}$, $f_S^{-1}(v_7) = \{v_{13}, v_{14}\}$ and $f_S^{-1}(v_{12}) = \{v_{10}, v_{11}\}$.

2.1 An ILP Model for the CAPMDS Problem

To formulate the CAPMDS problem in terms of an ILP, two types of binary variables are required. The original version of this ILP was published in [7]. Here we show (a slightly improved) version of the original ILP model with fewer binary variables. First, a binary variable x_i is associated to each vertex $v_i \in V$ indicating whether or not v_i is selected to be included in the solution. Secondly, for each edge $(v_i, v_j) \in E$ the model contains binary variables y_{ij} and y_{ji}. Hereby, variable y_{ij} takes values one iff vertex v_i dominates vertex v_j. In the same way, variable y_{ji} takes value one if v_j dominates v_i. The CAPMDS problem can thus be formulated as follows:

$$\textbf{minimize} \sum_{v_i \in V} x_i \tag{1}$$

$$\textbf{subject to} \sum_{v_j \in N(v_i)} y_{ji} \geq 1 - x_i \quad \forall v_i \in V \tag{2}$$

$$\sum_{v_j \in N(v_i)} y_{ij} \leq Cap(v_i) \quad \forall v_i \in V \tag{3}$$

$$y_{ij} \leq x_i \quad \forall v_i \in V, v_j \in N(v_i) \tag{4}$$

$$x_i, y_{ij} \in \{0, 1\} \tag{5}$$

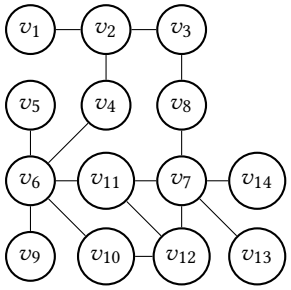
In the above formulation, constraint (2) ensures that all vertices that are not part of the solution have to be dominated by at least one neighbor, whereas constraint (3) specifies that the total number of vertices dominated by a given particular vertex v_i, is bounded by $Cap(v_i)$. Consequently, v_i can dominate at most $Cap(v_i)$ vertices from its (open) neighborhood.

3 THE CMSA FRAMEWORK

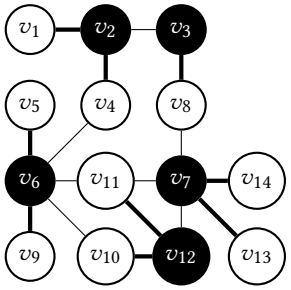
Construct, Merge, Solve & Adapt (CMSA) is an algorithmic framework for solving combinatorial optimization problems [2]. The primary and original goal of CMSA was to take profit from exact solvers in the context of problem instances that are too large in order to apply the exact solver directly. The application of this technique has shown excellent results, among others, for problems such as unbalanced minimum common string partition [1], project scheduling [17], maximum happy vertices [6], and test case generation for software checking [12].

A high-level description of our implementation of CMSA for the CAPMDS problem is provided in Algorithm 1. The algorithm maintains, at all times, an initially empty subset V' of the set of vertices V of the input graph. This set is henceforth called the *sub-instance*. Vertices are added to sub-instance V' by means of a probabilistic solution construction process which is implemented in function ProbabilisticSolutionGeneration(d_{rate}, l_{size}) (see line 8 of Algorithm 1). In particular, all vertices found in the solutions generated by this function are added to V'—if not already in V'—and their so-called *age value age*[] is initialized to zero (see lines 10 and 11 of Algorithm 1). Moreover, solving the sub-instance V' refers to the application of the ILP solver CPLEX in order to find—if possible within the imposed time limit of t_{solver} seconds—the optimal solution to sub-instance V', that is, the optimal CAPMDS solution in G that is limited to only contain vertices from V'. This is achieved by adding the following set of constraints to the ILP model from the previous section:

$$x_i = 0 \quad \forall x_i \in V \setminus V' \tag{6}$$



(a) CAPMDS problem instance, assuming a uniform capacity $Cap(v) = 2, \forall v \in V$.



(b) Minimum capacitated dominating set $S = \{v_2, v_3, v_6, v_7, v_{12}\}$.

Figure 1: An illustrative example of the CAPMDS problem. Note that the bold edges in (b) indicate the domination mapping. Vertices v_1 and v_4, for example, are mapped to vertex v_2 which forms part of the dominating set.

Algorithm 1 CMSA for the CAPMDS problem

```
1:  input: a problem instance (G, Cap)
2:  input: parameter values for age_max, n_a, t_solver, l_size, d_rate
3:  S_bsf := NULL
4:  V' := ∅
5:  age[v] := 0 for all v ∈ V
6:  while CPU time limit not reached do
7:    for i = 1, ..., n_a do
8:      S := ProbabilisticSolutionGeneration(d_rate, l_size)
9:      for all v ∈ S and v ∉ V' do
10:       age[v] := 0
11:       V' ← V' ∪ {v}
12:     end for
13:   end for
14:   S'_opt ← ApplyExactSolver(V', t_solver)
15:   if f(S'_opt) < f(S_bsf) then S_bsf := S'_opt
16:   Adapt(V', S'_opt, age_max)
17: end while
18: output: S_bsf
```

The process of solving the sub-instance V' is done in function ApplyExactSolver(V', t_{solver}); see line 14 of Algorithm 1. The algorithm takes as input the tackled problem instance (G, Cap), and values for a set of five required parameters: (1) the maximum allowed age of a vertex ($\text{age}_{\max}$), which establishes the number of iterations that a vertex is allowed to form part of sub-instance V' without appearing (as a dominating vertex) in the optimal solution to V'; (2) the parameter n_a, which defines the number of solutions generated in the construction phase of the algorithm at each iteration, that is, the number of calls to function ProbabilisticSolutionGeneration(d_{rate}, l_{size}); (3) the time limit (in seconds) used for running the ILP solver at each iteration (t_{solver}); (4) parameters l_{size} and d_{rate} that are used in function ProbabilisticSolutionGeneration(d_{rate}, l_{size}), which is described in more detail in the following subsection.

CMSA is also equipped with a mechanism for discarding seemingly useless solution components from the sub-instance V' at each iteration. In particular, first, the age values of all vertices in V' are incremented; second, the age values of all vertices in S'_{opt} are re-initialized to zero; and third, the vertices with age values greater than $\text{age}_{\max}$ are erased from V'. This mechanism is implemented in function Adapt(V', S'_{opt}, $\text{age}_{\max}$); see line 15 of Algorithm 1. This cleaning process has the aim of maintaining V' small enough in order to be able to solve the sub-instance to optimality (most of the times). If this is not possible, the output S'_{opt} of function Adapt(V', S'_{opt}, $\text{age}_{\max}$) is simply the best solution found by CPLEX within the allowed computation time.

The CMSA algorithm iterates while the CPU time limit is not reached. Moreover, it provides the best solution found during the search, S_{bsf}, as output.

3.1 Probabilistic Solution Generation

The function used for generating solutions in a probabilistic way (see line 8 of Algorithm 2) works as follows. It takes as input the problem instance (G, Cap), as well as values for the parameters l_{size} and d_{rate}. These two parameters are used for controlling the degree of stochasticity of the solution construction process. The process starts with an empty partial solution $S := \emptyset$. At each step, exactly one vertex from a set $W \subseteq V$ of vertices is added to S, until W is empty. Note that W is initially set to V. The question is how to choose a vertex v^* from W at each step of the procedure. First of all, vertices from W are evaluated by a dynamic greedy function $h()$ which depends on the current partial solution S. More specifically:

$$h(v) := \min\{Cap(v), \deg_W(v)\} \quad \forall v \in W \tag{7}$$

Hereby, $Cap(v)$ refers to the capacity of v (as outlined in Section 2) and $\deg_W(v) := |N(v) \cap W|$, that is, $\deg_W(v)$ is defined as the degree of v not considering nodes that are not in W. The $h()$-value of a vertex is henceforth called its heuristic value. The choice of a vertex v^* is then done as follows. First, a value $\delta \in [0, 1]$ is chosen uniformly at random. In case $\delta \leq d_{\text{rate}}$, v^* is chosen as the vertex that has the highest heuristic value among all vertices in W. Otherwise, a candidate list L containing the $\min\{l_{\text{size}}, |W|\}$ vertices from W with the highest heuristic values is generated, and v^* is chosen from L uniformly at random. Thus, the greediness of the solution construction procedure depends on the values of the determinism rate (d_{rate}) and the candidate list size (l_{size}). Note that when $\deg_W(v^*)$ is greater than $Cap(v^*)$, we have to choose which vertices from $N(v^*) \cap W$ vertex v^* is going to cover. In our current implementation, the set of $Cap(v^*)$ vertices is simply chosen from

Algorithm 2 Function ProbabilisticSolutionGeneration(d_{rate}, l_{size})

input: a problem instance (G, Cap)
input: parameter values for l_{size}, d_{rate}
$S := \emptyset$
$W := V$
while $W \neq \emptyset$ **do**
  Choose a random number $\delta \in [0, 1]$
  if $\delta \leq d_{\text{rate}}$ **then**
    Choose $v^* \in V$ such as $h_{v^*} \geq h_v$ for all $v \in W$
  else
    Let $L \subseteq W$ contain the $\min\{l_{\text{size}}, |W|\}$ vertices from W with the highest heuristic values
    Choose v^* uniformly at random from L
  end if
  $S := S \cup \{v^*\}$
  $W := W \setminus (C_{v^*} \cup \{v^*\})$
end while
output: S

$N(v^*) \cap W$ uniformly at random and denoted by C_{v^*}. Finally, W is updated at the end of each step. For this purpose, the selected vertex v^*, in addition to the vertices from C_{v^*}, are removed from W.

4 EXPERIMENTAL EVALUATION

The following three algorithmic approaches are compared on a large variety of benchmark instances: (1) The ILP solver IBM ILOG CPLEX v12.7 (henceforth simply called CPLEX) applied with default settings to the original problem instances, (2) the local search approach labelled LS_PD from [7], and (3) our implementation of CMSA for the CAPMDS problem. Remember that LS_PD is the current state of the art approach. Surprisingly, the performance of LS_PD has not yet been compared to the one of an ILP solver such as CPLEX. CPLEX and CMSA were implemented in ANSI C++ using GCC 5.4.0 for compiling the software under Ubuntu Linux 18.04. Moreover, both as a standalone solver and within CMSA, CPLEX was executed in one-threaded mode. The time limit for CPLEX and CMSA was 1000 seconds per run for all problem instances. All the experiments concerning CPLEX and CMSA were performed on machines with an Intel(R) Core(TM) i7-3770 processor with 3.4 GHz. The results of the experiments for LS_PD were taken from [7]. According to the authors, LS_PD was run on a machine with an Intel(R) Xeon(R) CPU E7-4830 processor with 2.13 GHz. However, the time limit applied to LS_PD was not stated in [7]. Instead they simply provided the average time when the best solution of a run was found.

4.1 Benchmark Instances

The performance of our CMSA algorithm was evaluated on all benchmark instances previously used for testing the earlier approaches for the CAPMDS problem; see also [14] and [7]. The corresponding benchmark set consists of two subsets. The first one contains 120 unit disk graphs (UDGs) created using the topology generator presented in [10]. In these instances, all vertices are randomly distributed over a Euclidean square of size 1000×1000. Moreover, the graphs are generated with two different range values: 150 and 200 units.[1] The second subset of instances consists of 180 general graphs taken from the set of so-called type I instances originally proposed by Shyu et *al.* [16]. The number of edges in these graphs, which depends on the number of vertices, ranges from 100 to 10000 edges. In both subsets, the number of nodes of the graphs is from $\{50, 100, 250, 500, 800, 1000\}$. Moreover, for each combination of the number of nodes and the range value concerning UDGs and for each combination of the number of nodes and the number of edges concerning general graphs, the benchmark set consists of 10 randomly generated instances. In fact, the results are presented in Section 4.3 in terms of averages over 10 instances. Finally, two types of capacities, namely uniform capacities and variable capacities were considered. In the case of uniform capacity, three different capacities of 2, 5 and α are tested, where α represents the average degree of the corresponding graph. In the case of variable capacity, the node capacities are randomly chosen from the following three intervals: $(2, 5)$, $(\alpha/5, \alpha/2)$ and $[1, \alpha]$. Note that the instance files come with the capacities already explicitly assigned.

4.2 Tuning of CMSA

The CMSA algorithm requires well-working values for five parameters: age_{max}, n_{a}, t_{solver}, l_{size} and d_{rate}. Please refer to Section 3 for a description of their function. The scientific parameter tuning tool irace [9] was used in order to determine the parameter values that were finally used for the experimental evaluation of the algorithm. For this purpose we split the instances into 36 groups, in the following way. All graphs with 50 or 100 vertices are classified as *small* (or *S*), all graphs with 250 or 500 vertices are classified as *medium* (or *M*), and all graphs with 800 or 1000 vertices are classified as *large* (or *L*). Moreover, a separate tuning process was performed for each graph type, each capacity and each capacity type. This makes a total of 36 tuning procedures. The parameter value domains for all these tuning runs were chosen as follows:

- $\text{age}_{\text{max}} \in \{1, 2, 3, 4, 10, 1000\}$
- $n_{\text{a}} \in \{1, 2, 5, 10, 30, 50\}$
- $t_{\text{solver}} \in \{3, 5, 10, 50, 75, 100\}$
- $l_{\text{size}} \in \{3, 5, 10, 20, 50\}$
- $d_{\text{rate}} \in \{0.3, 0.5, 0.7, 0.9\}$

Finally, notice that for each of the 36 tuning runs we used four instances generated for this purpose. The results of the tuning procedure are provided in Table 1.

4.3 Results

The CMSA algorithm was applied exactly once to each problem instance, with a computation time limit of 1000 seconds per run. The numerical results are presented in Tables 2, 3, 4a, and 4b. In particular, the results are presented in terms of averages over 10 problem instances in each table row. Each table provides the results of the three approaches mentioned at the beginning of this section: CPLEX, LS_PD, and CMSA. The first three table columns indicate the number of vertices in the graph (#Vertices), the number of edges (#Edges), respectively, the range value (Range) used to generate the graphs, and the capacity scheme (Capacity). The table columns

[1] Note that a higher range value results in more dense graphs.

Table 1: CMSA parameter values obtained with irace.

(a) UDG instances, variable cap.

Size	Cap.	d_{rate}	l_{size}	$\text{age}_{\max}$	n_{a}	t_{solver}
	$(2,5)$	0,5	50	5	10	50
S	$(\alpha/5, \alpha/2)$	0,5	20	3	50	10
	$[1,\alpha]$	0,9	10	10	5	50
	$(2,5)$	0,7	10	1000	50	50
M	$(\alpha/5, \alpha/2)$	0,7	3	1	5	75
	$[1,\alpha]$	0,5	50	1	1	10
	$(2,5)$	0,5	50	2	10	75
L	$(\alpha/5, \alpha/2)$	0,9	5	1	1	100
	$[1,\alpha]$	0,3	50	1	2	75

(b) General graphs, variable cap.

Size	Cap.	d_{rate}	l_{size}	$\text{age}_{\max}$	n_{a}	t_{solver}
	$(2,5)$	0,3	50	10	50	100
S	$(\alpha/5, \alpha/2)$	0,5	3	5	2	75
	$[1,\alpha]$	0,5	20	1000	10	3
	$(2,5)$	0,5	10	5	50	100
M	$(\alpha/5, \alpha/2)$	0,9	50	2	50	100
	$[1,\alpha]$	0,5	50	1	1	75
	$(2,5)$	0,3	50	1	2	100
L	$(\alpha/5, \alpha/2)$	0,9	10	1	1	75
	$[1,\alpha]$	0,5	10	1	1	50

(c) UDG instances, uniform cap.

Size	Cap.	d_{rate}	l_{size}	$\text{age}_{\max}$	n_{a}	t_{solver}
	2	0,3	50	5	10	50
S	5	0,3	10	5	1	5
	α	0,7	10	5	2	75
	2	0,7	5	5	1	3
M	5	0,7	10	3	1	100
	α	0,7	3	3	1	3
	2	0,5	20	1000	2	50
L	5	0,3	10	1000	1	50
	α	0,5	5	3	2	50

(d) General graphs, uniform cap.

Size	Cap.	d_{rate}	l_{size}	$\text{age}_{\max}$	n_{a}	t_{solver}
	2	0,3	10	1000	10	3
S	5	0,3	10	1000	10	3
	α	0,7	50	3	5	100
	2	0,5	3	10	5	10
M	5	0,3	5	1	1	75
	α	0,9	10	1	1	75
	2	0,9	50	3	2	5
L	5	0,9	10	2	1	75
	α	0,7	10	1	1	50

4–6 present the information of CPLEX in terms of the average solution quality, the average computation time at which the best solutions were found, and the average optimality gap (in percent). Columns 7–8 provide the information of LS_PD in terms of the average solution quality and the average computation time at which the best solutions were found. Finally, columns 9–10 provide the same information for CMSA.

The experimental results allow to make the following observations:

- First of all, the good performance of CPLEX for the considered instances indicates that earlier papers on this problem should have tested the application of ILP solvers. In particular, CPLEX can solve all 10 problem instances to optimality in 44 out of 54 cases concerning general graphs with uniform capacity, in 43 out of 54 cases concerning general graphs with variable capacity, in 29 out of 36 cases concerning UDG instances with uniform capacities, and in 25 out of 36 cases concerning UDG instances with variable capacities. Problem difficulty seems to increase for CPLEX with a growing vertex capacity and with a growing graph size (in terms of the number of vertices). Moreover, variable capacity graphs seem to be slightly (but consistently) more difficult for CPLEX.
- In particular, CPLEX clearly outperforms the current state-of-the-art algorithm LS_PD. This is with the exception of some of the cases concerning very large graphs. See, for example, general graphs with uniform capacity on 100 vertices, 10000 edges, and a capacity of α. An even more notable example concerns the case of the UDG graphs with variable capacity (capacities from $[1,\alpha]$) on 1000 vertices. The performance of CPLEX degrades strongly in these cases (given the computation time limit of 1000 seconds).
- The best-performing algorithm is CMSA. It generally matches the results of CPLEX in those cases where CPLEX performs very well. Moreover, its performance does not degrade in the case of the largest problem instances. In fact, in only 18 out of 180 cases, the performance of CMSA is slightly worse than the one of the best-performing algorithm in these cases. Concerning computation time, we can observe that CMSA is generally very fast. However, in some cases the algorithm requires much more computation time, which is an aspect that must be studied in future work.

5 CONCLUSIONS AND FUTURE WORK

This paper has dealt with an NP-Hard optimization problem in graphs, the so-called minimum capacitated dominating set (CAP-MDS) problem. For solving this problem, we proposed an implementation of a hybrid algorithm known as Construct, Merge, Solve & Adapt (CMSA). The results show that not only CMSA, but also the standalone ILP solver CPLEX, are able to outperform the current state-of-the-art algorithm from the literature. However, while the performance of CPLEX starts to degrade in the context of the largest problem instances, this is not the case for CMSA.

Concerning future work, we plan to investigate the performance of CMSA for even larger problem instances. Moreover, we plan to investigate the use of alternative greedy guiding functions for the probabilistic construction of solutions. Finally, we aim to study why CMSA takes much more computation time in some cases.

ACKNOWLEDGMENTS

Pedro Pinacho-Davidson acknowledges the support of the CONICYT PCI Program under grant REDESI170160.

REFERENCES

[1] Christian Blum. 2016. Construct, Merge, Solve and Adapt: Application to unbalanced minimum common string partition. In *Proceedings of HM 2016 – 10th International Workshop on Hybrid Metaheuristics (Lecture Notes in Computer Science)*, Maria J. Blesa, Christian Blum, Angelo Cangelosi, Vincenzo Cutello, Alessandro Di Nuovo, Mario Pavone, and El-Ghazali Talbi (Eds.), Vol. 9668. Springer International Publishing, 17–31.

[2] Christian Blum, Pedro Pinacho, Manuel López-Ibánez, and José A. Lozano. 2016. Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization. *Computers & Operations Research* 68 (2016), 75 – 88. https://doi.org/10.1016/j.cor.2015.10.014

[3] Marek Cygan, Marcin Pilipczuk, and Jakub Onufry Wojtaszczyk. 2010. Capacitated Domination Faster Than O(2n). In *Algorithm Theory - SWAT 2010*, Haim Kaplan (Ed.). Lecture Notes in Computer Science, Vol. 6139. Springer-Verlag, Berlin, Heidelberg, 74–80.

[4] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA.

[5] Fabian Kuhn and Thomas Moscibroda. 2010. Distributed approximation of capacitated dominating sets. *Theory of Computing Systems* 47, 4 (2010), 811–836.

[6] Rhyd Lewis, Dhananjay Thiruvady, and Kerri Morgan. 2019. Finding happiness: An analysis of the maximum happy vertices problem. *Computers & Operations Research* 103 (2019), 265–276.

[7] Ruizhi Li, Shuli Hu, Peng Zhao, Yupeng Zhou, and Minghao Yin. 2018. A novel local search algorithm for the minimum capacitated dominating set. *Journal of the Operational Research Society* 69, 6 (2018), 849–863.

[8] Mathieu Liedloff, Ioan Todinca, and Yngve Villanger. 2010. Solving Capacitated Dominating Set by Using Covering by Subsets and Maximum Matching. In *Graph Theoretic Concepts in Computer Science*, Dimitrios M. Thilikos (Ed.). Lecture Notes in Computer Science, Vol. 6410. Springer-Verlag, Berlin, Heidelberg, 88–99.

[9] Manuel López-Ibánez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic

Table 2: Results for general graphs with uniform capacity.

#Vertices	#Edges	Capacity	CPLEX	CPLEX$_{time}$	CPLEX$_{gap}$	LS_PD	LS_PD$_{time}$	CMSA	CMSA$_{time}$
50	100	2	**17.0**	<0.1	0.0	**17.0**	0.0	**17.0**	<0.1
50	250	2	**17.0**	<0.1	0.0	**17.0**	0.0	**17.0**	<0.1
50	500	2	**17.0**	0.1	0.0	**17.0**	0.0	**17.0**	<0.1
100	100	2	**34.0**	0.2	0.0	**34.0**	0.0	**34.0**	<0.1
100	250	2	**34.0**	0.1	0.0	**34.0**	0.0	**34.0**	<0.1
100	500	2	**34.0**	0.1	0.0	**34.0**	0.0	**34.0**	0.1
250	250	2	**84.0**	1.7	0.0	**84.0**	3.6	**84.0**	0.3
250	500	2	**84.0**	0.4	0.0	**84.0**	5.7	**84.0**	0.4
250	1000	2	**84.0**	0.6	0.0	**84.0**	10.0	**84.0**	0.4
500	500	2	**167.0**	10.8	0.0	168.6	0.0	**167.0**	1.2
500	1000	2	**167.0**	1.7	0.0	170.2	55.5	**167.0**	1.8
500	2000	2	**167.0**	1.7	0.0	167.5	38.4	**167.0**	1.4
800	1000	2	**267.0**	2.7	0.0	274.2	160.9	**267.0**	3.6
800	2000	2	**267.0**	3.4	0.0	272.7	127.1	**267.0**	3.9
800	5000	2	**267.0**	4.6	0.0	267.8	44.4	**267.0**	3.2
1000	1000	2	**334.0**	145.8	0.0	338.4	136.5	**334.0**	7.9
1000	5000	2	**334.0**	5.7	0.0	336.0	126.0	**334.0**	6.5
1000	10000	2	**334.0**	12.6	0.0	**334.0**	4.2	**334.0**	5.8
Avg			**150.5**	10.7		151.9	39.6	**150.5**	2.0
50	100	5	**11.9**	0.1	0.0	**11.9**	0.0	**11.9**	0.2
50	250	5	**9.0**	0.2	0.0	**9.0**	0.0	**9.0**	<0.1
50	500	5	**9.0**	0.2	0.0	**9.0**	0.0	**9.0**	<0.1
100	100	5	**33.6**	<0.1	0.0	**33.6**	0.2	**33.6**	<0.1
100	250	5	**20.0**	0.9	0.0	20.3	5.9	**20.0**	70.7
100	500	5	**17.0**	0.6	0.0	**17.0**	0.1	**17.0**	0.3
250	250	5	**83.3**	0.2	0.0	83.5	11.6	**83.3**	0.3
250	500	5	**57.8**	9.7	0.0	59.9	46.0	**57.8**	28.4
250	1000	5	**42.0**	15.7	0.0	45.0	32.6	**42.0**	93.9
500	500	5	**167.0**	0.9	0.0	168.3	10.7	**167.0**	1.7
500	1000	5	**114.9**	821.4	1.2	121.5	80.5	115.1	301.8
500	2000	5	84.1	254.1	0.2	92.2	71.3	**84.0**	336.0
800	1000	5	**242.5**	5.8	0.0	253.2	101.8	243.1	205.6
800	2000	5	164.8	1000.0	9.4	176.2	223.6	**162.8**	574.7
800	5000	5	**134.0**	69.3	0.0	140.4	131.6	**134.0**	4.7
1000	1000	5	**333.7**	3.1	0.0	338.7	75.7	**333.7**	10.5
1000	5000	5	**167.0**	205.6	0.0	181.0	140.3	**167.0**	40.8
1000	10000	5	**167.0**	149.7	0.0	171.0	62.7	**167.0**	3.7
Avg			103.3	141.0		107.3	55.3	**103.2**	92.6
50	100	α	**12.0**	0.2	0.0	**12.0**	0.0	**12.0**	0.4
50	250	α	**6.0**	0.5	0.0	**6.0**	0.1	**6.0**	9.2
50	500	α	**3.8**	0.9	0.0	**3.8**	0.0	**3.8**	0.1
100	100	α	**34.0**	0.2	0.0	**34.0**	0.0	**34.0**	<0.1
100	250	α	**20.0**	1.0	0.0	20.2	5.9	20.1	83.6
100	500	α	**12.2**	5.5	0.0	**12.2**	2.6	**12.2**	30.3
250	250	α	**84.0**	2.1	0.0	**84.0**	2.1	**84.0**	0.5
250	500	α	**58.3**	30.6	0.0	61.7	17.8	**58.3**	110.5
250	1000	α	**36.7**	999.8	10.0	37.9	41.1	**36.7**	162.9
500	500	α	**167.0**	15.8	0.0	168.4	32.3	**167.0**	2.8
500	1000	α	**116.2**	966.2	2.9	126.4	59.0	116.8	257.6
500	2000	α	75.3	1000.0	17.4	78.6	101.0	**73.7**	665.2
800	1000	α	**267.0**	3.6	0.0	274.0	120.8	**267.0**	4.6
800	2000	α	164.9	1001.0	9.4	178.1	159.2	**162.8**	537.3
800	5000	α	91.1	993.6	30.2	92.2	329.4	93.0	717.9
1000	1000	α	**334.0**	131.7	0.0	338.4	34.7	**334.0**	13.7
1000	5000	α	132.5	951.3	25.9	137.4	352.2	135.0	782.9
1000	10000	α	87.6	993.9	42.1	**81.3**	604.8	86.8	518.7
Avg			**94.6**	394.3		97.0	103.5	**94.6**	216.6

algorithm configuration. *Operations Research Perspectives* 3 (2016), 43 – 58. https://doi.org/10.1016/j.orp.2016.09.002

[10] Michele Mastrogiovanni. 2007. The Clustering Simulation Framework: A Simple Manual. http://www.michele-matrogiovanni.net/software/download/README.pdf. [Online; accessed 2017].

[11] Thomas Moscibroda. 2007. Clustering. In *Algorithms for Sensor and Ad Hoc Networks: Advanced Lectures*, Dorothea Wagner and Roger Wattenhofer (Eds.). Springer-Verlag, Berlin, Heidelberg, 37–61.

[12] Jose Antonio Ortega-Toro, Javier Ferrer, and Francisco Chicano. 2018. CMSA para el problema de la generación de casos de prueba priorizados en líneas de productos software. In *Jornadas de Ingeniería del Software y Bases de Datos*. Sociedad de Ingeniería de Software y Tecnologías de Desarrollo de Software (SISTEDES). http://hdl.handle.net/11705/JISBD/2018/091

[13] Anupama Potluri and Alok Singh. 2012. A Greedy Heuristic and Its Variants for Minimum Capacitated Dominating Set. In *Contemporary Computing*, Manish et *al* Parashar (Ed.). Communications in Computer and Information Science, Vol. 306. Springer-Verlag, Berlin, Heidelberg, 28–39.

Table 3: Results for general graphs with variable capacity.

#Vertices	#Edges	Capacity	CPLEX	CPLEX$_{time}$	CPLEX$_{gap}$	LS_PD	LS_PD$_{time}$	CMSA	CMSA$_{time}$
50	100	$(2, 5)$	**13.0**	0.1	0.0	**13.0**	0.2	**13.0**	0.2
50	250	$(2, 5)$	**9.0**	0.1	0.0	**9.0**	0.0	**9.0**	0.1
50	500	$(2, 5)$	**9.0**	0.1	0.0	**9.0**	0.0	**9.0**	<0.1
100	100	$(2, 5)$	**33.7**	0.1	0.0	**33.7**	0.1	**33.7**	0.1
100	250	$(2, 5)$	**21.9**	0.7	0.0	22.3	8.9	**21.9**	0.5
100	500	$(2, 5)$	**17.0**	0.4	0.0	17.2	6.5	**17.0**	0.8
250	250	$(2, 5)$	**83.7**	0.7	0.0	**83.7**	6.6	**83.7**	0.3
250	500	$(2, 5)$	**63.2**	11.9	0.0	66.5	20.7	63.3	11.8
250	1000	$(2, 5)$	43.7	177.1	0.3	48.4	20.6	**43.6**	110.8
500	500	$(2, 5)$	**167.0**	4.5	0.0	168.2	60.4	**167.0**	2.1
500	1000	$(2, 5)$	125.5	702.0	1.0	135.2	104.5	**125.4**	448.2
500	2000	$(2, 5)$	88.2	725.3	3.0	98.9	63.2	**87.2**	544.5
800	1000	$(2, 5)$	**248.1**	23.1	0.0	261.7	180.3	248.2	79.2
800	2000	$(2, 5)$	**181.2**	1000.0	4.9	199.1	174.9	181.5	341.7
800	5000	$(2, 5)$	**134.1**	34.9	0.0	144.5	161.5	**134.1**	28.1
1000	1000	$(2, 5)$	**333.8**	31.0	0.0	338.6	139.9	**333.8**	3.9
1000	5000	$(2, 5)$	**169.0**	123.7	0.0	189.4	175.7	**169.0**	85.1
1000	10000	$(2, 5)$	**167.0**	94.8	0.0	172.2	207.2	**167.0**	27.5
Avg			**106.0**	162.8		111.7	74.0	**106.0**	93.6
50	100	$(\alpha/5, \alpha/2)$	**17.7**	<0.1	0.0	**17.7**	0.4	**17.7**	0.1
50	250	$(\alpha/5, \alpha/2)$	**9.0**	0.1	0.0	**9.0**	0.2	**9.0**	0.1
50	500	$(\alpha/5, \alpha/2)$	**5.0**	0.4	0.0	**5.0**	0.0	**5.0**	0.1
100	100	$(\alpha/5, \alpha/2)$	**50.0**	<0.1	0.0	**50.0**	0.1	**50.0**	<0.1
100	250	$(\alpha/5, \alpha/2)$	**34.3**	0.1	0.0	34.9	6.7	**34.3**	0.1
100	500	$(\alpha/5, \alpha/2)$	**17.0**	0.4	0.0	17.3	6.4	**17.0**	1.0
250	250	$(\alpha/5, \alpha/2)$	**125.0**	0.2	0.0	**125.0**	8.9	**125.0**	0.5
250	500	$(\alpha/5, \alpha/2)$	**86.8**	0.4	0.0	91.4	19.3	**86.8**	0.5
250	1000	$(\alpha/5, \alpha/2)$	**51.4**	1.0	0.0	54.9	20.2	**51.4**	3.0
500	500	$(\alpha/5, \alpha/2)$	**250.0**	0.8	0.0	251.1	61.2	**250.0**	1.7
500	1000	$(\alpha/5, \alpha/2)$	**172.9**	1.4	0.0	183.5	36.5	**172.9**	1.9
500	2000	$(\alpha/5, \alpha/2)$	**101.3**	5.7	0.0	110.8	59.8	**101.3**	18.7
800	1000	$(\alpha/5, \alpha/2)$	**400.0**	2.1	0.0	401.6	85.9	**400.0**	2.6
800	2000	$(\alpha/5, \alpha/2)$	**273.4**	3.1	0.0	292.2	107.6	**273.4**	6.5
800	5000	$(\alpha/5, \alpha/2)$	**115.0**	74.3	0.0	127.0	101.8	115.1	178.6
1000	1000	$(\alpha/5, \alpha/2)$	**500.0**	3.0	0.0	505.6	108.9	**500.0**	8.6
1000	5000	$(\alpha/5, \alpha/2)$	**168.1**	131.6	0.0	188.0	118.4	**168.1**	77.4
1000	10000	$(\alpha/5, \alpha/2)$	105.7	841.6	13.6	**104.7**	123.0	107.1	247.9
Avg			**137.9**	59.2		142.8	48.1	138.0	30.5
50	100	$[1, \alpha]$	**13.7**	0.1	0.0	**13.7**	0.8	**13.7**	0.2
50	250	$[1, \alpha]$	**7.2**	0.4	0.0	**7.2**	0.6	**7.2**	3.0
50	500	$[1, \alpha]$	**3.9**	1.2	0.0	**3.9**	0.0	**3.9**	0.4
100	100	$[1, \alpha]$	**40.1**	0.1	0.0	**40.1**	0.0	**40.1**	<0.1
100	250	$[1, \alpha]$	**23.3**	0.7	0.0	23.7	7.4	**23.3**	3.6
100	500	$[1, \alpha]$	**13.5**	5.7	0.0	14.1	5.9	13.6	23.4
250	250	$[1, \alpha]$	**98.9**	0.3	0.0	99.0	6.6	**98.9**	0.3
250	500	$[1, \alpha]$	**67.3**	3.4	0.0	71.5	18.7	**67.4**	63.6
250	1000	$[1, \alpha]$	40.5	948.8	5.3	44.6	27.3	**40.2**	205.4
500	500	$[1, \alpha]$	**202.7**	0.9	0.0	203.2	28.3	**202.7**	1.3
500	1000	$[1, \alpha]$	**135.9**	51.8	0.0	147.7	55.0	136.3	365.5
500	2000	$[1, \alpha]$	83.1	1000.0	10.8	92.6	99.8	**81.4**	505.4
800	1000	$[1, \alpha]$	**300.2**	2.1	0.0	310.5	90.1	**300.2**	4.0
800	2000	$[1, \alpha]$	186.6	1000.0	3.5	208.2	124.8	**186.2**	442.6
800	5000	$[1, \alpha]$	100.3	985.7	21.9	104.8	227.2	**98.1**	683.7
1000	1000	$[1, \alpha]$	**400.8**	3.6	0.0	405.0	146.0	**400.8**	6.4
1000	5000	$[1, \alpha]$	149.5	949.9	20.0	156.4	335.5	**143.8**	866.9
1000	10000	$[1, \alpha]$	132.9	1000.0	54.2	92.4	387.0	**90.1**	541.8
Avg			111.1	330.9		113.3	86.7	**108.2**	194.6

[14] Anupama Potluri and Alok Singh. 2013. Metaheuristic algorithms for computing capacitated dominating set with uniform and variable capacities. *Swarm and Evolutionary Computation* 13 (2013), 22–33.

[15] Rajmohan Rajaraman. 2002. Topology control and routing in ad hoc networks: A survey. *ACM SIGACT News* 33, 2 (2002), 60–73.

[16] Shyong Jian Shyu, Peng-Yeng Yin, and Bertrand MT Lin. 2004. An ant colony optimization algorithm for the minimum weight vertex cover problem. *Annals of Operations Research* 131, 1-4 (2004), 283–304.

[17] Dhananjay Thiruvady, Christian Blum, and Andreas T. Ernst. 2019. Maximising the Net Present Value of Project Schedules Using CMSA and Parallel ACO. In *Proceedings of HM 2019 – 11th International Workshop on Hybrid Metaheuristics (Lecture Notes in Computer Science)*, Maria J. Blesa, Christian Blum, Haroldo Gambini Santos, Pedro Pinacho-Davidson, and Julio Godoy del Campo (Eds.), Vol. 11299. Springer, 16–30.

[18] Jane Yang Yu and Peter Han Joo Chong. 2005. A survey of clustering schemes for mobile ad hoc networks. *IEEE Communications Surveys & Tutorials* 7, 1 (2005), 32–48.

Table 4: Results for the UDG instances.

(a) Uniform capacity instances

#Vertices	Range	Capacity	CPLEX	CPLEX$_{time}$	CPLEX$_{gap}$	LS_PD	LS_PD$_{time}$	CMSA	CMSA$_{time}$
50	150	2	**17.2**	<0.1	0.0	**17.2**	0.1	**17.2**	0.1
50	200	2	**17.0**	0.1	0.0	**17.0**	0.0	**17.0**	<0.1
100	150	2	**34.0**	0.1	0.0	34.5	0.1	**34.0**	0.1
100	200	2	**34.0**	0.2	0.0	34.1	0.0	**34.0**	0.1
250	150	2	**84.0**	0.8	0.0	86.0	0.0	**84.0**	0.3
250	200	2	**84.0**	0.9	0.0	85.4	0.0	**84.0**	0.3
500	150	2	**167.0**	3.1	0.0	171.6	0.2	**167.0**	1.1
500	200	2	**167.0**	4.4	0.0	170.3	0.1	**167.0**	1.5
800	150	2	**267.0**	8.0	0.0	273.9	0.2	**267.0**	6.6
800	200	2	**267.0**	11.7	0.0	272.4	0.1	**267.0**	8.2
1000	150	2	**334.0**	13.9	0.0	342.6	0.0	**334.0**	10.3
1000	200	2	**334.0**	19.1	0.0	340.4	0.0	**334.0**	16.9
Avg			**150.5**	5.2		153.8	0.1	**150.5**	3.8
50	150	5	**12.9**	<0.1	0.0	**12.9**	0.1	**12.9**	0.1
50	200	5	**10.0**	0.1	0.0	**10.0**	0.0	**10.0**	0.2
100	150	5	**18.7**	0.3	0.0	18.8	0.0	**18.7**	1.2
100	200	5	**17.4**	0.4	0.0	**17.4**	0.1	**17.4**	0.2
250	150	5	**42.0**	3.5	0.0	43.7	0.4	**42.0**	0.3
250	200	5	**42.0**	3.5	0.0	43.0	0.0	**42.0**	0.4
500	150	5	**84.0**	25.7	0.0	86.2	0.2	**84.0**	1.2
500	200	5	**84.0**	50.7	0.0	85.0	0.0	**84.0**	1.4
800	150	5	**134.0**	215.9	0.0	137.0	1.0	**134.0**	3.8
800	200	5	**134.0**	387.9	0.0	135.7	0.1	**134.0**	4.6
1000	150	5	**167.0**	459.6	0.0	171.0	2.7	**167.0**	6.1
1000	200	5	173.3	787.6	2.9	169.6	0.0	**167.0**	7.5
Avg			76.6	161.3		77.5	0.4	**76.1**	2.2
50	150	α	**14.4**	<0.1	0.0	**14.4**	79.4	**14.4**	0.1
50	200	α	**10.0**	0.1	0.0	**10.0**	22.5	**10.0**	0.2
100	150	α	**18.7**	0.3	0.0	18.9	0.0	**18.7**	1.9
100	200	α	**11.0**	0.4	0.0	**11.0**	0.0	**11.0**	0.7
250	150	α	**18.5**	4.5	0.0	19.4	8.3	18.6	18.8
250	200	α	**11.3**	6.3	0.0	11.5	1.7	11.4	10.2
500	150	α	**18.6**	164.1	0.0	20.7	250.1	18.8	143.9
500	200	α	**11.3**	90.9	0.0	12.1	129.5	11.7	213.0
800	150	α	**19.4**	906.2	4.9	21.3	319.7	19.5	278.8
800	200	α	12.3	733.4	7.4	12.6	21.4	**12.0**	196.1
1000	150	α	25.8	942.4	22.5	21.5	2.6	**19.7**	421.2
1000	200	α	14.7	953.4	23.5	12.9	0.0	**12.0**	341.9
Avg			15.5	316.8		15.5	69.6	**14.8**	135.6

(b) Variable capacity instances

#Vertices	Range	Capacity	CPLEX	CPLEX$_{time}$	CPLEX$_{gap}$	LS_PD	LS_PD$_{time}$	CMSA	$CMSA_time$
50	150	$(2, 5)$	**14.5**	<0.1	0.0	**14.5**	0.0	**14.5**	<0.1
50	200	$(2, 5)$	**11.1**	0.1	0.0	**11.1**	0.0	**11.1**	<0.1
100	150	$(2, 5)$	**21.8**	1.1	0.0	**21.8**	3.5	**21.8**	0.6
100	200	$(2, 5)$	**17.6**	0.4	0.0	17.8	3.7	**17.6**	0.4
250	150	$(2, 5)$	**42.1**	2.6	0.0	43.0	18.4	**42.1**	1.4
250	200	$(2, 5)$	**42.0**	2.3	0.0	**42.0**	36.0	**42.0**	1.3
500	150	$(2, 5)$	**84.0**	11.7	0.0	84.9	35.7	**84.0**	5.2
500	200	$(2, 5)$	**84.0**	15.2	0.0	**84.0**	33.4	**84.0**	6.3
800	150	$(2, 5)$	**134.0**	55.4	0.0	134.9	251.8	**134.0**	7.6
800	200	$(2, 5)$	**134.0**	71.1	0.0	134.1	595.3	**134.0**	9.9
1000	150	$(2, 5)$	**167.0**	105.1	0.0	168.6	235.4	**167.0**	15.3
1000	200	$(2, 5)$	**167.0**	119.0	0.0	167.5	408.1	**167.0**	21.9
Avg			**76.6**	32.0		77.0	135.1	**76.6**	5.8
50	150	$(\alpha/5, \alpha/2)$	**25.4**	0.1	0.0	**25.4**	0.0	**25.4**	<0.1
50	200	$(\alpha/5, \alpha/2)$	**17.3**	<0.1	0.0	**17.3**	0.1	**17.3**	<0.1
100	150	$(\alpha/5, \alpha/2)$	**29.9**	0.2	0.0	30.3	6.7	**29.9**	0.2
100	200	$(\alpha/5, \alpha/2)$	**17.8**	0.4	0.0	**17.8**	3.4	**17.8**	0.2
250	150	$(\alpha/5, \alpha/2)$	**31.6**	4.4	0.0	32.4	25.2	**31.6**	0.9
250	200	$(\alpha/5, \alpha/2)$	**18.9**	26.3	0.0	19.2	13.3	**18.9**	1.4
500	150	$(\alpha/5, \alpha/2)$	**32.0**	661.5	0.0	32.9	35.9	**32.0**	16.4
500	200	$(\alpha/5, \alpha/2)$	79.6	602.3	74.7	**19.4**	116.3	**19.4**	11.1
800	150	$(\alpha/5, \alpha/2)$	732.0	936.5	96.5	33.3	91.8	**31.7**	21.9
800	200	$(\alpha/5, \alpha/2)$	796.6	1000.0	100.0	19.8	100.1	**19.6**	10.0
1000	150	$(\alpha/5, \alpha/2)$	997.8	1000.0	100.0	33.5	173.6	**32.5**	17.5
1000	200	$(\alpha/5, \alpha/2)$	996.0	1000.0	100.0	20.0	88.2	**19.2**	16.9
Avg			314.6	436.4		25.1	54.6	**24.6**	8.0
50	150	$[1, \alpha]$	**16.7**	<0.1	0.0	**16.7**	0.1	**16.7**	0.2
50	200	$[1, \alpha]$	**12.1**	0.1	0.0	**12.1**	0.5	**12.1**	0.1
100	150	$[1, \alpha]$	**21.4**	0.3	0.0	**21.4**	4.3	**21.4**	1.4
100	200	$[1, \alpha]$	**12.8**	1.5	0.0	13.0	0.8	**12.8**	6.5
250	150	$[1, \alpha]$	**20.8**	38.0	0.0	21.9	6.1	**20.8**	41.5
250	200	$[1, \alpha]$	**12.8**	58.2	0.0	13.0	5.5	**12.8**	19.5
500	150	$[1, \alpha]$	21.9	940.6	8.3	22.6	55.7	**21.1**	122.2
500	200	$[1, \alpha]$	13.2	926.1	11.2	13.0	53.0	**12.5**	199.4
800	150	$[1, \alpha]$	22.7	973.2	14.1	22.7	163.0	**21.1**	515.6
800	200	$[1, \alpha]$	726.9	997.4	98.1	13.2	76.5	**12.7**	298.9
1000	150	$[1, \alpha]$	909.2	1000.0	97.9	22.6	500.4	**21.3**	606.3
1000	200	$[1, \alpha]$	1000.0	1000.0	100.0	13.0	319.3	**12.6**	400.5
Avg			232.5	494.8		17.1	98.8	**16.5**	184.3